(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
12 September 2002 (12.09.2002)

PCT

(10) International Publication Number
WO 02/071757 A2

(51) International Patent Classification[7]: H04N 7/24

(21) International Application Number: PCT/GB02/00881

(22) International Filing Date: 28 February 2002 (28.02.2002)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0105518.5 7 March 2001 (07.03.2001) GB
0128995.8 4 December 2001 (04.12.2001) GB

(71) Applicant *(for all designated States except US)*: **INTERNET PRO VIDEO LIMITED** [GB/GB]; Mount Pleasant House, 2 Mount Pleasant, Huntingdon Road, Cambridge CB3 ORN (GB).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **KING, Tony, Richard** [GB/GB]; 28 Marlowe Road, Newnham, Cambridge CB3 9JW (GB).

(74) Agent: **LANGLEY, Peter, James**; Origin Limited, 52 Muswell Hill Road, London N10 3JR (GB).

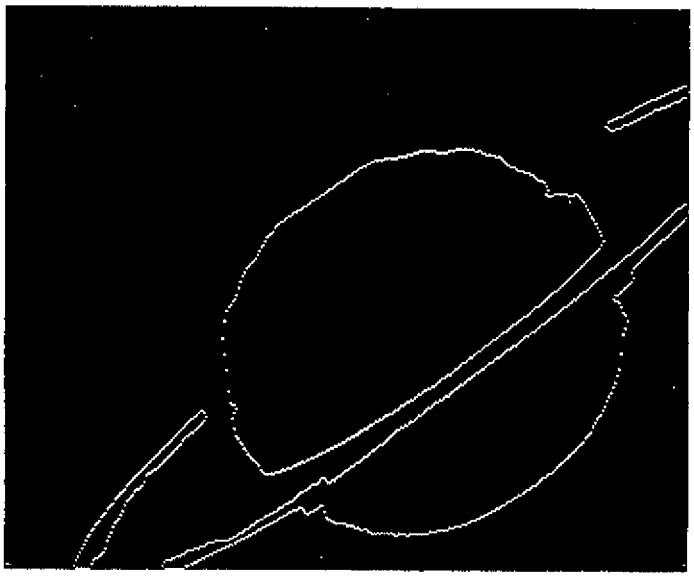
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A METHOD OF PROCESSING VIDEO INTO AN ENCODED BITSTREAM



(57) Abstract: In a method of processing video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device, the processing of the video results in the bitstream (a) representing the video in a vector graphic format with quality labels which are device independent, and also (b) being decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

WO 02/071757 A2

A method of processing video into an encoded bitstream

Technical Field

This invention relates to a method of a method of processing video into an encoded bitstream. This may occur when processing pictures or video into instructions in a vector graphics format for use by a limited-resource display device.

Background Art

Systems for the manipulation and delivery of pictures or video in a scalable form allow the client for the material to request a quality setting that is appropriate to the task in hand, or to the capability of the delivery or decoding system. Then, by storing a representation at a particular quality in local memory, such systems allow the client to refine that representation over time in order to gain extra quality. Conventionally, such systems take the following approach: an encoding of the media is obtained by applying an algorithm whose parameters (e.g. quantisation level) are set to some "coarse" level. The result is a bitstream which can be decoded and the media fully reconstructed, although at a reduced quality with respect to the original. Subsequent encodings of the input are then obtained with progressively "better quality" parameter settings, and these can be combined with the earlier encodings in order to obtain a reconstruction to any desired quality.

Such a system may include a method for processing the image data into a compressed and layered form where the layers provide a means of obtaining and decoding data over time to build up the quality of the image. An example is described in PCT/GB00/01614 to Telemedia Limited. Here the progressive nature of the wavelet encoding in scale-space is used in conjunction with a ranking of wavelet coefficients in significance order, to obtain a bitstream that is Scalable in many dimensions.

Such systems, however, make assumptions about the capabilities of the client device, in particular, as regards the display hardware, where the ability to render multi-bit pixel values into a framestore at video update rates, is usually necessary. At the extreme end of the mobile computing spectrum however, multi-bit deep framestores may not be available, or if they are, the constraints of limited connection capacity, CPU, memory, and battery life, make the rendering of even the lowest quality video a severe drain on resources. In order to

address this problem a method of adapting the data to the capability of the client device is required. This is a hard problem in the context of video which is conventionally represented in a device-dependent low-level way, as intensity values with a fixed number of bits sampled on a rectangular grid. Typically, in order to adapt to local constraints, such material would have to be completely decoded and then reprocessed into a more suitable form.

A more flexible media format would describe the picture in a higher-level, more generic, and device-independent way, allowing efficient processing into any of a wide range of display formats. In the field of computer graphics, vector formats are well known and have been in use since images first appeared on computer screens. These formats typically represent the pictures as strokes, polygons, curves, filled areas, and so on, and as such make use of a higher-level and wider range of descriptive elements than is possible with the standard image pixel-format. An example of such a vector file format is Scalable Vector Graphics (SVG). If images can be processed into vector format while retaining (or even enhancing) the meaning or sense of the image, and instructions for drawing these vectors can be transmitted to the device rather than the pixel values (or transforms thereof), then the connection, CPU and rendering requirements potentially can all be dramatically reduced.

Summary of the Invention

In a first aspect, there is provided a method of processing video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device; wherein the processing of the video results in the bitstream:

(a) representing the video in a vector graphic format with quality labels which are device independent, and

(b) being decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

The quality labels may enable scalable reconstruction of the video at the device and also at different devices with different display capabilities. The method is particularly useful in devices which are resource constrained, such as mobile telephones and handheld computers.

The following steps may occur as part of processing the video into a vector graphics format with quality labels:

(a) describing the video in terms of vector based graphics primitives;

(b) grouping these graphics primitives into features;

(c) assigning to the graphics primitives and/or to the features values of perceptual significance;

(d) deriving quality labels from these values of perceptual significance.

An image, represented in the conventional way as intensity samples on a rectangular grid, can be converted into a graphical form and represented as an encoding of a set of shapes. This encoding represents the image at a coarse scale but with edge information preserved. It also serves as a base level image from which further, higher quality, encodings, are generated using one or more encoding methods. In one implementation, video is encoded using a hierarchy of video compression algorithms, where each algorithm is particularly suited to the generation of encoded video at a given quality level.

In a second aspect, there is a method of decoding video which has been processed into an encoded bitstream in which the encoded bitstream has been be sent over a WAN to device;

wherein the decoding of the bitstream involves (i) extracting quality labels which are device independent and (ii) enabling the device to display a vector graphics based representation of the video at a quality determined by the quality labels, so that the quality of the video displayed on the device is determined by the resource constraints of the device.

In a third aspect, there is an apparatus for encoding video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device; wherein the apparatus is capable of processing the video into the bitstream such that the bitstream:

(a) represents the video in a vector graphic format with quality labels which are device independent, and

(b) is decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

In a fourth aspect, there is a device for decoding video which has been processed into an encoded bitstream in which the encoded bitstream has been be sent over a WAN to the device;

wherein the device is capable of decoding the bitstream by (i) extracting quality labels which are device independent and (ii) displaying a vector graphics based representation of the video at a quality determined by the quality labels, so that the quality of the video displayed on the device is determined by the resource constraints of the device.

In a fifth and final aspect, there is a video file bitstream which has been encoded by a process comprising the steps of processing an original video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device; wherein the processing of the video results in the encoded bitstream:

(a) representing the video in a vector graphic format with quality labels which are device independent, and

(b) being decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

Briefly, an implementation of the invention works as follows:

A grey-scale image is converted to a set of regions. In a preferred embodiment, the set of regions corresponds to a set of binary images such that each binary image represents the original image thresholded at a particular value. A number of quantisation levels *max_levels* is chosen and the histogram of the input image is equalised for that number of levels, i.e., each quantisation level is associated with an equal number of pixels. Threshold values t(1), t(2),..., t(*max_levels*), where t is a value between the minimum and maximum value of the grey-scale, are derived from the equalisation step and used to quantize the image into *max_levels* binary images consisting of foreground regions (1) and background (0). For each of the *max_levels* image levels the following steps are taken: The regions are grown in order to fill small holes and so eliminate some 'noise'. Then, to ensure that no 'gaps' open up in the regions during detection of their perimeters, any 8-fold connectivity of the background within a foreground

region is removed, and 8-fold connected foreground regions are thickened to a minimum of 3-pixel width.

In another embodiment, the regions are found using a "Morphological Scale-Space Processor"; a non-linear image processing technique that uses shape analysis and manipulation to process multidimensional signals such as images. The output from such a processor typically consists of a succession of images containing regions with increasingly larger-scale detail. These regions may represent recognisable features of the image at increasing scales and can conveniently be represented in a scale-space tree, in which nodes hold region information (position, shape, colour) at a given scale, and edges represent scale-space behavior (how coarse-scale regions are formed from many fine-scale ones).

These regions may be processed into a description (the shape description) that describes the shape, colour, position, visual priority, and any other aspect, of the regions, in a compact manner. This description is processed to provide feature information, where a feature is an observable characteristic of the image. This information may include any of the following: the sign of the intensity gradient of the feature (i.e., whether the contour represents the perimeter of a filled region or a hole), the average intensity of the feature, and the 'importance' of the feature, as represented by this contour.

In a preferred embodiment, the perimeters of the regions are found, unique labels assigned to each contour, and each labelled contour processed into a list of coordinates. For each of the *max_levels* image levels, and for each contour within that level it is established whether the contour represents a boundary or a hole using a scan-line parity-check routine (Theo Pavlidis "Algorithms for Graphics and Image Processing", Springer-Verlag, P.174). Then a grey-scale intensity is estimated and assigned to this contour by averaging the grey-scale intensities around the contour.

Finally, the contours are grouped into features by sorting the contours into families of related contours, and each feature is assigned a perceptual significance computed from the intensity gradients of the feature. Also, each contour within the feature is individually assigned a perceptual significance computed from the intensity gradient in the locality of the contour. Quality labels are then derived from the values of perceptual significance for both the contours and features in order to enable determination of position in a quality hierarchy.

The contour coordinates may be sorted in order to put the coordinates in pixel adjacency order in order that, in the fitting step, the correct curves are modeled.

In the preferred embodiment of this aspect of the invention, the contour is split into a set of simplified curves that are single-valued functions of the independent variable x, i.e., the curves do not double-back on themselves, so a point with ordinate x is adjacent to a point with ordinate x+1.

Parametric curves may then be fitted to the contours.

In a preferred embodiment, a piecewise cubic Bezier curve fitting algorithm is used as described in: Andrew S. Glassner (ed), Graphics Gems Volume 1, P612, "An Algorithm for Automatically Fitting Digitised Curves". The curves are priority-ordered to form a list of graphics instructions in a vector graphics format that allow a representation of the original image to be reconstructed at a client device.

For each level, starting with the lowest, and for each contour representing a filled region, the curve is written to file in SVG format. Then, for each level starting with the highest, and for each contour representing a hole, the curve written to file in SVG format. This procedure adapts the well-known "painters algorithm" in order to obtain the correct visual priority for the regions. The SVG client renders the regions in the order in which they are written in the file: by rendering regions of increasing intensity order "back-to-front" and then rendering regions of decreasing intensity order "front-to-back" the desired approximation to the input image is reconstructed.

The region description may be transmitted to a client which decodes and reconstructs the video frames to a "base" quality level. A second encoding algorithm is then employed to generate enhancement information that improves the quality of the reconstructed image.

In a preferred embodiment, the segmented and vectorised image is reconstituted at the encoder at a resolution equivalent to the "root" quadrant of a quadtree decomposition. This is used as an approximation to, or predictor for, the true root data values. The encoder subtracts the predicted, from the true root quadrant, encodes the difference using an entropy encoding scheme, and transmits the result. The decoder performs the inverse function, adding the root difference to the reconstructed root, and using this as the start point in the inverse transform.

Brief Description of Figures

Note:- in the figures, the language used in the code fragments is MATLAB m-code.

Figure 1 shows a code fragment for the 'makecontours' function.

Figure 2 shows a code fragment for the 'contourtype' function.

Figure 3 shows a code fragment for the 'contourcols' function.

Figure 4 shows a code fragment for the 'contourassoc' function.

Figure 5 shows a code fragment for the 'contourgrad' function.

Figure 6 shows a code fragment for the 'adjorder' function.

Figure 7 shows a code fragment for the 'writebezier' function.

Figure 8 shows a flow chart representing the process of grouping contours into features.

Figure 9 shows a flow chart representing the process of assigning values of perceptual significance to features and contous.

Figure 10 shows a flow chart representing the process of assigning quality labels to contours.

Figure 11 shows a diagram of the data structures used.

Figure 12 shows the original monochrome 'Saturn' image.

Figures 13 - 16 show the contours at levels 1 - 4, respectively.

Figure 17 shows the contours at all levels superimposed.

Figure 18 shows the rendered SVG image.

Figure 19 shows a scalable encoder.

Figure 20 shows a scalable decoder.

Best Mode for Carrying out the Invention

Key Concepts

Scalable Vector Graphics

An example of a scalable vector file format is Scalable Vector Graphics (Scalable Vector Graphics (SVG) 1.0 Specification, W3C Candidate Recommendation, 2 August 2000). SVG is a proposed standard format for vector graphics which is a namespace of XML and which is designed to work well across platforms, output resolutions, color spaces, and a range of available bandwidths. SVG.

Wavelet Transform

The wavelet transform has only relatively recently matured as a tool for image analysis and compression. Reference may for example be made to Mallat, Stephane G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.11, No.7, pp 674-692 (Jul 1989) in which the Fast Wavelet Transform (FWT) is described. The FWT generates a hierarchy of power-of-two images or subbands where at each step the spatial sampling frequency - the 'fineness' of detail which is represented - is reduced by a factor of two in x and y. This procedure decorrelates the image samples with the result that most of the energy is compacted into a small number of high-magnitude coefficients within a subband, the rest being mainly zero or low-value, offering considerable opportunity for compression.

Each subband describes the image in terms of a particular combination of spatial/frequency components. At the base of the hierarchy is one subband - the root - which carries the average intensity information for the image, and is a low-pass filtered version of the input image. This subband can be used in Scalable image transmission systems as a coarse-scale approximation to the input image, which, however, suffers from blurring and poor edge definition.

Scale-Space Filtering

The idea of scale-space was developed for use in computer vision investigations and is described in, for example, AP Witkin: Scale space filtering - A new approach to multi-scale description, Ullman, Richards (Eds.), Image Understanding, Ablex, Norwood, NJ, 79-95,

1984. In a multi-scale representation, structures at coarse scales represent simplifications of the corresponding structures at finer scales. A multi-scale representation of an image can be obtained by the wavelet transform, as described above, or convolution using a Gaussian kernel. However, such linear filters result in a blurring of edges at coarse scales, as in the case of the wavelet root quadrant, as described above.

Browse Quality

In certain applications, the ability quickly to gain a sense of structure and movement outweighs the need to render a picture as accurately as possible. Such a situation occurs when a human user of a video delivery system wishes to find a particular event in a video sequence, for example, during an editing session; here the priority is not to appreciate the image as an approximation to reality, but to find out what is happening in order to make a decision. In such situations a stylised, simplified, or cartoon-like representation is as useful as, and arguably better than, an accurate one, as long as the higher-quality version is available when required.

Segmentation

In order to obtain a scale-space representation that simplifies or removes detail whilst preserving edge definition, a different approach must be taken to the problem of image simplification. Segmentation is the process of identifying and labelling regions that are "similar", according to some relation. A segmented image replaces smooth gradations in intensity with sharply defined areas of constant intensity but preserves perceptually significant features, and retains the essential structure of the image. A simple and straightforward approach to doing this involves applying a series of thresholds to the image pixels to obtain constant intensity regions, and sorting these regions according to their scale (obtained by counting interior pixels, or other geometrical methods which take account of the size and shape of the perimeter). These regions, typically, will correlate poorly with perceptually significant features in the original image, but can still represent the original in a stylised way.

To obtain a better correlation between image features and segmented regions non-linear image processing techniques can be employed as described in, for example, P. Salembier and J. Serra. "Flat zones filtering, connected operators and filters by reconstruction", IEEE Transactions on Image Processing, 3(8):1153-1160, August 1995, which describes a Morphological segmentation technique.

Morphological segmentation is a shape-based image processing scheme that uses connected operators (operators that transform local neighbourhoods of pixels) to remove and merge regions such that intra-region similarity tends to increase and inter-region similarity tends to decrease. This results in an image consisting of so-called "flat zones": regions with a particular colour and scale. Most importantly, the edges of these flat zones are well-defined and correspond to edges in the original image.

A specific embodiment of the invention will now be described by way of example.

Conversion of input image to set of binary images representing regions

Referring to the code fragment of **figure 1**, a number of quantisation levels *max_levels* is chosen and the histogram of the input image is equalised for that number of levels. The equalisation transform matrix is then used to derive a vector of threshold values and this vector is used to quantise the image into *max_levels* levels. The histogram of the resulting quantised image is flat (i.e. each quantisation level is associated with an equal number of pixels). Then, for each of the *max_levels* levels, the image is thresholded at level L to convert to a binary image, consisting of foreground regions (1) and background (0).

Conversion of binary images to coordinate lists representing contours

Referring again to the code fragment of **figure 1**, for each of the *max_levels* binary images the following steps are taken: The regions are grown in order to fill small holes and so eliminate some 'noise'. The 'grow' operation involves setting a pixel to '1' if five or more pixels in the 3-by-3 neighbourhood are '1's; otherwise it is set to '0'.

Then, to ensure that no gaps open up in the regions during subsequent processing, any 8-fold connectivity of the background is removed using a diagonal fill, and 8-fold connected foreground regions are widened to a minimum 3-pixel span using a thicken operation that

adds pixels to the exterior of regions. The perimeters of the resulting regions are located and a new binary image created with pixels set to represent the perimeters. Each set of 8-connected pixels is then located and overwritten with a unique label. Then every connected set of pixels with a particular label is found and a list of pixel coordinates is built.

Determination of contour colour and type

Referring to the code fragment of **figure 2**, for each of the *max_levels* image levels, and for each contour within that level it is established whether the contour represents a fill or a hole at this level using a scan-line parity-check routine (Theo Pavlidis "Algorithms for Graphics and Image Processing", Springer-Verlag, P.174). Then, referring to the code fragment of **figure 3**, for each contour a grey-scale intensity is estimated and assigned to this contour by averaging the grey-scale intensities around the contour.

Feature extraction and quality labelling from contours

The contours are grouped into features where each feature is assigned a perceptual significance computed from the intensity gradients of the feature. Also, each contour within the feature is individually assigned a perceptual significance computed from the intensity gradient in the locality of the contour. This is done as follows. Referring to the code fragment of **figure 4** and the flow-chart of **figure 8**: starting with the highest-intensity fill-contour (rather than hole-contour), each contour at level L is associated with the contour at level L-1 that immediately encloses it, again using scan-line parity-checking. An association list is built that relates every contour to its 'parent' contour so that groups of contours representing a feature can be identified. The feature is assigned an ID and a reference to the contour list is made in a feature table. The process is then repeated for hole-contours, starting with the one with the lowest-intensity.

Referring to the code fragment of **figure 5** and the flow-chart of **figure 9**, perceptual significances are then assigned to features and contours in the following way. Starting with the highest-intensity fill-contour of a feature, and at each of a fixed number of positions (termed the *fall-lines*) around this contour, the intensity gradient is calculated by determining the distance to the parent contour. These gradients are median-filtered and averaged and the value thus obtained - *pscontour* - gives a reasonable indication of perceptual significance of the contour. The association list is used to descend through all the rest of the enclosing

contours. Then the gradients down each of the *fall-lines* of all the contours for the feature are calculated, median-filtered and averaged, and the value thus obtained - *psfeature* - gives a reasonable indication of perceptual significance of the feature as a whole.

The final step is to derive quality labels from the values of perceptual significance for the contours and features in order to enable determination of position in a quality hierarchy. Referring to the flow-chart of **figure 10**, quality labels are initialised as the duple {Ql, Qg} (local and global quality) on each contour descriptor. The features are sorted with respect to *psfeature*. The first (most significant) feature is found and all of the contour descriptors in its list have their Ql set to 1; then the next most significant feature is found and the contour descriptors have their Ql set to 2, and so on. Thus, all the contours within a feature have the same value of Ql; contours belonging to different features have different values of Ql.

As a second step all the contours are sorted with respect to *pscontour*, and linearly increasing values of Qg, starting with 1, are written to their descriptors. Thus, every contour in the scene has a unique value of Qg.

Two orderings of the data are thus obtained using the quality labels: Ql ranks localised image features into significance order, Qg ranks contours into global significance order. This allows a decoder to choose the manner in which a picture is reconstructed: whether to bias in favour of reconstructing individual local features with the best fidelity first, or obtaining a global approximation to the entire scene first.

The diagram of **figure 11** outlines the data structures used when assigning quality labels to contours. The feature indicated comprises three contours. Local and global gradients are computed using the eight fall-lines shown and the values for *psfeature, pscontour, Qg and Ql* are written in the tables.

Reordering and filtering of contours

After the previous operations have been completed the coordinates in each list are in scan-order, i.e., the order in which they were detected. In order for curve-fitting to work they need to be re-ordered such that each coordinate represents a pixel adjacent to its immediate 8-fold connected neighbour. Referring to the code fragment of **figure 6** - of the independent variable, i.e., that never change direction with respect to increasing this is done as follows: The contour may be complicated, with many changes of direction, but it cannot

cross itself, or have multiple paths. The algorithm splits the contour into a list of simpler curves that are single-valued functions scan number (or x-value). On these curves each value of the independent variable x maps to just one point, so points at x(n) and x(n+1) must be adjacent. The start and finish points of these curves are found, then for each curve these points are tested against all others to determine which curve connects to which other(s). Finally, the curves are traversed in connection order to generate the list of pixel coordinates in adjacency order. As part of the reordering process, runs of pixels on the same scan line are detected and replaced by a single point to reduce the size of data handed on to the fitting process.

Bezier curve fitting

The piecewise cubic Bezier curve fitting algorithm used in the preferred embodiment of the invention is described in: Andrew S. Glassner (ed), Graphics Gems Volume 1, P612, "An Algorithm for Automatically Fitting Digitised Curves".

Visual priority ordering

Referring to the code fragment of **figure 7**, for each level starting with the lowest, and for each contour representing a filled region, the curve is written to file in SVG format. Then, for each level starting with the highest, and for each contour representing a hole, the curve written to file in SVG format. This procedure adapts the well-known "painters algorithm" in order to obtain the correct visual priority for the regions. The SVG client renders the regions in the order in which they are written in the file: by rendering regions of increasing intensity order "back-to-front" and then rendering regions of decreasing intensity order "front-to-back" the desired approximation to the input image is reconstructed.

Scalable encoding using a vector graphics base level encoding

Referring to the diagrams of a scalable encoder and decoder (**figures 15** and **16**), at the encoder the input image is segmented, shape-encoded, converted to vector graphics and transmitted as a low-bitrate base level image; it is also rendered at the wavelet root quadrant resolution and used as a predictor for the root quadrant data. The error in this prediction is entropy-encoded and transmitted together with the compressed wavelet detail coefficients. This compression may be based on the principle of spatially oriented trees, as described in PCT/GB00/01614 to Telemedia Limited. The decoder performs the inverse function; it

renders the root image and presents this as a base level image; it also adds this image to the root difference to obtain the true root quadrant data which is then used as the start point for the inverse wavelet transform.

Industrial Applicability

As a simple example of the use of the invention consider the situation in which it is desired that material residing on a picture repository be made available to a range of portable devices with displays with an assortment of spatial and grey-scale resolution - possibly some with black-and-white output only. Using the methods of the current invention the material is processed into a single file in SVG format. The devices are loaded with SVG viewer software that allows reconstruction of picture data irrespective of the capability of the individual client device.

Claims

1. A method of processing video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device; wherein the processing of the video results in the bitstream:

(a) representing the video in a vector graphic format with quality labels which are device independent, and

(b) being decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

2. The method of Claim 1 in which the quality labels enable scalable reconstruction of the video at the device and also at different devices with different display capabilities.

3. The method of Claim 1 in which the following steps occur as part of processing the video into a vector graphics format with quality labels:

(a) describing the video in terms of vector based graphics primitives;

(b) grouping these graphics primitives into features;

(c) assigning to the graphics primitives and/or to the features values of perceptual significance;

(d) deriving quality labels from these values of perceptual significance.

4. The method of Claim 1 in which multiple processing steps are applied to the video, with each processing step producing an encoded bitstream with different quality characteristics.

5. The method of Claim 3 in which the vector based graphics primitives are selected from the group comprising:

(a) straight lines or

(b) curves.

6. The method of Claim 3 in which the values of perceptual significance relate to one or more of the following:

(a) individual local features;

(b) a global approximation to an entire scene in the video.

7. The method of Claim 3 in which the values of perceptual significance relate to one or more of the following:

(a) sharpness of an edge

(b) size of an edge

(c) type of shape

(d) colour consistency.

8. The method of Claim 1 in which the video is an image and/or an image sequence.

9. The method of Claim 3 where the video constitutes the base level in a scalable image delivery system, and where the features represented by graphics primitives in the video have a simplified or stylised appearance, and have well defined edges.

10. The method of Claim 9 where the image processing involves converting a grey-scale image into a set of binary images obtained by thresholding.

11. The method of Claim 9 where the processing involves converting a grey-scale image into a set of regions obtained using morphological processing.

12. The method of Claim 9 or 10, where the processing further involves the steps of region processing to eliminate detail, perimeter determination, and processing into a coordinate list.

13. The method of Claim 12 where the processing further involves the generation of perceptual significance information for both the graphics primitives and features, that are used to derive quality labels, that enable determination of position in a quality hierarchy.

14. The method of Claim 13 where the processing further involves re-ordering of the list such that each coordinate represents a pixel adjacent to its immediate 8-fold connected neighbour

15. The method of Claim 14 where the processing further involves fitting parametric curves to the contours.

16. The method of Claim 15 where the processing further involves priority-ordering the contour curves representing filled regions front-to-back, and contour curves representing holes back-to-front, in order to form a list of graphics instructions in a vector graphics format that allow a representation of the original image to be reconstructed at a client device.

17. A method of decoding video which has been processed into an encoded bitstream in which the encoded bitstream has been be sent over a WAN to device;

wherein the decoding of the bitstream involves (i) extracting quality labels which are device independent and (ii) enabling the device to display a vector graphics based representation of the video at a quality determined by the quality labels, so that the quality of the video displayed on the device is determined by the resource constraints of the device.

18. An apparatus for encoding video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device; wherein the apparatus is capable of processing the video into the bitstream such that the bitstream:

(a) represents the video in a vector graphic format with quality labels which are device independent, and

(b) is decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

19. A device for decoding video which has been processed into an encoded bitstream in which the encoded bitstream has been be sent over a WAN to the device;

wherein the device is capable of decoding the bitstream by (i) extracting quality labels which are device independent and (ii) displaying a vector graphics based representation of

the video at a quality determined by the quality labels, so that the quality of the video displayed on the device is determined by the resource constraints of the device.

20. A video file bitstream which has been encoded by a process comprising the steps of processing an original video into an encoded bitstream in which the encoded bitstream is intended to be sent over a WAN to a device; wherein the processing of the video results in the encoded bitstream:

(a) representing the video in a vector graphic format with quality labels which are device independent, and

(b) being decodable at the device to display, at a quality determined by the resource constraints of the device, a vector graphics based representation of the video.

```
A method of function  makecontours(image, max_levels, max_labels, min_component_len,
thresh)
%
% Equalise the histogram of the input image.
% Quantize the image at the predetermined thresholds to give binary image set.
% Process to remove spurious detail, and to ensure minimum region thickness.
% Find perimeter pixels and assign labels.
%
[image, eqtrans] = histeq(image, max_levels) ;
[nelements, thresh] = hist(eqtrans, max_levels) ;
thresh = round(thresh*255) ;

idx_contour = 1 ;                                   % contour counter

for level = 1:max_levels,
        img = (image>thresh(level)) ;               % convert to binary image
        img = bwmorph(img, 'majority', 16) ;        % fill holes
        img = bwmorph(img, 'diag') ;                % remove 8-connectivity of
background
        img = bwmorph(img, 'thicken') ;                 % minimum 3 pixel thick areas
        img = bwperim(img) ;

        [im_labelled, nlabels] = bwlabel(img, 8) ;      % Find & label 8-connected
pixels

        % Find coordinates of labelled components
        %
        ncomponents = min([nlabels, max_labels]) ;      % max no. contours allowed
        ncontours(level) = 0 ;                      % count of contours per level

        for ic = 1:ncomponents,
                [lrow, lcol] = find(im_labelled==ic) ;  % Coordinate list
                points = [lcol, lrow] ;
                if (size(points, 1) > min_component_len)        % Discard small regions

                        cntr_points(idx_contour) = {points} ; % contour coordinates
                        cntr_label(idx_contour) = (ic) ;        % label value for this contour

                        idx_contour = idx_contour + 1 ;
                        ncontours(level) = ncontours(level) + 1 ;
                end
        end
end
```

Figure 1

```
function contourtype(cntr_points, max_levels, ncontours)
%
% Determine if closed contour is boundary (1) or hole (0): adapted from scanline
% contour filling algorithm using parity check (Pavlidis P.174).
%
% Find any point on this curve - result applies to curve as a whole;
% Choose the first point in the list as the candidate point (xtest, ytest);
% For every other curve that intersects y with x<xtest, determine edge parity;
% Handle tangent/multiple pixels properly using contour-line adjacency graph;
% If sum of parities is even then contour is boundary, otherwise hole.
%
contourbase = 1 ;

for level = 1:max_levels,
        for ic = 1:ncontours(level),
                contour = cntr_points{contourbase + ic - 1} ;
                paritytotal = 0 ;
                othercontours = find((1:ncontours(level))~=ic) ;
                for curv = othercontours,
                        testcontour = cntr_points{contourbase + curv - 1} ;

                        [parity, errorflag] = ip_parityFind(testcontour, contour) ;
                        paritytotal = paritytotal + parity ;
                end

                if (rem(paritytotal, 2)==0)
                        cntr_type(contourbase + ic - 1) = 1 ;
                else
                        cntr_type(contourbase + ic - 1) = 0 ;
                end
        end
        contourbase = contourbase + ncontours(level) ;
end
```

Figure 2

```
function contourcols(cntr_points, cntr_type, image)

for ic = 1:length(cntr_points),
        %
        % Determine the contour colour by averaging the pixel intensities
        % around the contour.
        %
        points = cntr_points{ic} ;
        siz = size(image) ;
        inds = sub2ind(siz, points(:,2), points(:,1)) ;
        cntr_cols(ic) = round(mean(image(inds))) ;
end
```

Figure 3

```
function contourassoc(cntr_points, cntr_type, ncontours)
%
% For each contour at level L find the contour at level L-1
% that immediately encloses it, using scan-line parity-checking.
% Build an association list that relates every contour to its 'parent'
%
idx_contour = 1 ;
conts = idx_contour:(idx_contour + ncontours(1) - 1) ;          % contours at currrent
level
conts = conts(find(cntr_type(conts))) ;                         % find fill-contours only
seq = 1 ;

for (ic = conts),

        cntr_assoc(ic) = 0 ;                                    % zero denotes lowest contour
assoc
        seq = seq + 1 ;
end
idx_contour    = idx_contour + ncontours(1) ;

for (il = 2:length(ncontours)),
        prevconts = conts ;                                     % contours at level L-1
        conts = idx_contour:(idx_contour + ncontours(il) - 1) ;     % contours at level L
        conts = conts(find(cntr_type(conts))) ;                 % find fill-contours only
        for (ip = prevconts),
                seq = 1 ;
                for (ic = conts),
                        [parity, errorflag] = ip_parityFind(cntr_points{ip}, cntr_points{ic}) ;
                        if (rem(parity, 2)~=0)
                                if (~isempty(cntr_assoc(ip)))
                                        cntr_assoc(ic) = ip ;
                                        seq = seq + 1 ;
                                end
                        end
                end
        end
        idx_contour = idx_contour + ncontours(il) ;
end
```

Figure 4

```
function contourgrad(cntr_points, cntr_assoc, ncontours, ngradients)
%
% Use the association list to identify groups of contours that represent features.
%
% At each of a fixed number of positions around the max contour calculate the
% intensity gradient at a tangent to the contour by determining the distance
% to the parent contour.
% Descend through, and process, all the enclosing contours.
% Find the next-highest unprocessed intensity contour and repeat until all
% the contours have been processed.
% Associate the contour groups with feature IDs in the cntr_feature list.
%
cntr_grad = {} ;
featureID = 1 ;
conts = fliplr(find(cntr_assoc)) ;                    % contour list, starting with
highest intensity
while (~(isempty(conts)))
        ic = conts(1) ;
        cntr = cntr_points{ic} ;
        stride = fix(length(cntr)/ngradients) ;       % Space beteen sample points
        gradindex = 1 ;
        for (grad = 1:ngradients)
                %
                % Choose the next point around the contour
                %
                point = cntr(gradindex, :) ;
                thisc = ic ;
                parentc = cntr_assoc(thisc) ;         % the parent contour id
                ig = 1 ;
                gradients = [] ;
                while (parentc~=0)
                        %
                        % Starting at the current point, go down the intensity fall-line
                        % updating a vector of gradients as we go.
                        %
                        parentcntr = cntr_points{parentc} ;   % the parent contour points

                        xdiffs = abs(parentcntr(:, 1)-point(1)) ;
                        ydiffs = abs(parentcntr(:, 2)-point(2)) ;
                        diffs = xdiffs + ydiffs ;     % manhatten distance measure
                        mindiffs = min(diffs) ;       % shortest path to parent
                        gradients(ig) = mindiffs ;

                        minidx = find(diffs==mindiffs) ;      % find intersection of fall line
parent
                        point = parentcntr(minidx(1), :) ;    % with parent contour
```

```
                    if (~isempty(conts))            % remove current contour from
process list
                        conts = conts(find(conts~=thisc)) ;
                    end
                    thisc = parentc ;                       % descend the fall line
                    parentc = cntr_assoc(thisc) ;
                    ig = ig + 1 ;
                end
                cntr_grad(featureID, grad) = {gradients} ;      % update feature list
                gradindex = gradindex + stride ;
            end
            cntr_feature(featureID) = ic ;                  % 'maxima' contour for this feature
            featureID = featureID + 1 ;
end
```

Figure 5

```
function  adjorder(points)
%
% All input points must be 8-connected.
% There must be a single path (no stubs, intersections, etc).
%
        error = 0 ;
        npoints = size(points, 1) ;

        firstscan = min(points(:,1)) ;
        lastscan = max(points(:,1)) ;
        scans = [firstscan:lastscan] ;
        nscans = length(scans) ;
        miny = min(points(:,2)) ;
        maxy = max(points(:,2)) ;
        linemapl = sparse(nscans, 1) ;
        linemaph = sparse(nscans, 1) ;

        nextcurve = 1 ;

        % Construct a scanline-ordered cell array of intersections with the curve.
        % We also remove multiple linear points on a scan.
        %
        for sc = 1:nscans,

                % Split up the contour into separate curves that are linearly
```

```
% related to the scanline (i.e. one sample point per scan).
% Bulid a linemap matrix (row=scanline, col=curve number) with entries
% min and max row values per scanline per curve ID.
%
% Find start points of all runs of connected pixels; subtract the
% list of row values from a shifted version of itself and look for
% discontinuities (i.e. steps greater than one).
%
runs = [points(find(points(:,1)==scans(sc)), 2)] ;
nsamples = length(runs) ;
shruns = [runs(1); runs(1:nsamples-1)] ;
runstarts = find(abs(runs - shruns)>1) ;

runends = [runstarts; nsamples+1] ;                    % end points
of all runs
runstarts = [1; runstarts] ;                           % start points
of all runs
lenruns = runends - runstarts ;                        % lengths of all runs
nruns = length(runstarts) ;                            % number of
runs

foundflag = 0 ;

if (sc==1)                  % start off the linemap matrix
        p = 1 ;
        for j = 1:nruns,
                run = runs(p:p+lenruns(j)-1) ;
                linemapl(sc, j) = min(run) ;
                linemaph(sc, j) = max(run) ;
                p = p + lenruns(j) ;

                nextcurve = nextcurve + 1 ;
        end
else                        % connect pixels to existing curves, else start new
curves
        lastsegsl = linemapl(sc-1, :) ;
        lastsegsh = linemaph(sc-1, :) ;
        nzsegs = find(lastsegsl) ;
        p = 1 ;
        for j = 1:nruns,
                run = runs(p:p+lenruns(j)-1) ;
                bl = min(run) ;
                bh = max(run) ;
                for k = nzsegs,
                        al = lastsegsl(k) ;
                        ah = lastsegsh(k) ;
```

```
                                % See if the new run connects to one of the existing
curves.
                                % If so, add to the existing curve, else create a new
one.
                                %
                                if (((bl<=ah+1) & (bl>=al)) | ((bh>=al-1) &
(bh<=ah))) ;
                                        linemapl(sc, k) = bl ;
                                        linemaph(sc, k) = bh ;
                                        foundflag = 1 ;

                                        % Run has been matched with curve, so
remove curve from list
                                        % and stop looking.
                                        %
                                        nzsegs = nzsegs(find(nzsegs~=k)) ;
                                        break ;
                                end
                        end
                        if (~foundflag) % create a new line
                                linemapl(sc, nextcurve) = bl ;
                                linemaph(sc, nextcurve) = bh ;

                                nextcurve = nextcurve + 1 ;
                        else
                                foundflag = 0 ;
                        end
                        p = p + lenruns(j) ;
                end
        end
end

% Find the endpoints of the curves
%
for (idxcrv = 1:size(linemapl, 2)),
        curve = find(linemapl(:, idxcrv)) ;

        startscan = curve(1) ;
        endscan = curve(length(curve)) ;
        endpt(idxcrv, 1) = startscan ;
        endpt(idxcrv, 2) = endscan  ;
        endpt(idxcrv, 3) = linemapl(startscan, idxcrv ) ;
        endpt(idxcrv, 4) = linemaph(startscan, idxcrv ) ;
        endpt(idxcrv, 5) = linemapl(endscan, idxcrv ) ;
        endpt(idxcrv, 6) = linemaph(endscan, idxcrv ) ;
end
```

```
% Find connected curves.
% Offsets of (+-1,+-1), & (+-1,0) give connectivity.
%
ncurves = size(endpt, 1) ;
curveref = zeros(ncurves, 2) ;          % list of connecting curve numbers

starts = endpt(:, 1) ;
ends = endpt(:, 2) ;

% deltapoints is a ncurves-by-4 matrix:
% [startscans-1, endscans-1, startscans+1, endscans+1; ....]
% It lists all the points that result when the start/end scan
% numbers are displaced by +-1 pixel.
%
deltapoints = [endpt(:, 1:2)-1, endpt(:, 1:2)+1] ;
findstarts = [] ;
findends = [] ;

for (idxcrv = 1:ncurves),
        % Find curves that connect; first, find curves whose scanline start/end.
        % points are within +- 1 line of those of the current curve.
        %
        % The 'start' is defined as the terminal with the smaller
        % scan line value; the 'end' as that with the greater.
        %
        % a & b matrices have same dimensions as deltapoints with a '1' signifying
        % a match with the current start/end.
        % findstarts & findends list the curve numbers for the scanline matches
        % for the start and end of the current curve, respectively,
        % (taking care to exclude the curve we are matching against).
        %
        a = reshape(starts(idxcrv)==deltapoints(:), ncurves, 4) ;
        fs = find((a(:,1) | a(:,2)) | (a(:,3) | a(:,4))) ;
        if (~isempty(fs))
                findstarts = (fs(find(fs~=idxcrv)))' ;
        end

        b = reshape(ends(idxcrv)==deltapoints(:), ncurves, 4) ;
        fe = find((b(:,1) | b(:,2)) | (b(:,3) | b(:,4))) ;
        if (~isempty(fe))
                findends = (fe(find(fe~=idxcrv)))' ;
        end

        % Now match the scan intersection start/ends of the current curve
        % against those of the candidates found in the scanline match.
        % When a match is found enter its curve ID into the curveref table.
```

```
            %
            currintersectstart = [(endpt(idxcrv, 3)-1):(endpt(idxcrv, 3)+1), (endpt(idxcrv,
4)-1):(endpt(idxcrv, 4)+1)] ;
            currintersectend = [(endpt(idxcrv, 5)-1):(endpt(idxcrv, 5)+1), (endpt(idxcrv,
6)-1):(endpt(idxcrv, 6)+1)] ;
            currdeltastart = [deltapoints(idxcrv, 1):deltapoints(idxcrv, 3)] ;
            currdeltaend = [deltapoints(idxcrv, 2):deltapoints(idxcrv, 4)] ;

            % Handle special case where curve is single run on scanline
            %
            if ((all(currdeltastart==currdeltaend)) &
(all(currintersectstart==currintersectend)))
                  currintersectstart = endpt(idxcrv, 3)-1 ;
                  currintersectend = endpt(idxcrv, 6)+1 ;
            end

            for (s = findstarts),          % try to match the start of the curve
                  if ((any(ismember(currintersectstart, endpt(s, 3:4)))) &...
                              (any(ismember(currdeltastart, endpt(s, 1)))))
                        curveref(idxcrv, 1) = s ;
                        break ;
                  elseif ((any(ismember(currintersectstart, endpt(s, 5:6)))) &...
                              (any(ismember(currdeltastart, endpt(s, 2)))))
                        curveref(idxcrv, 1) = s ;
                        break ;
                  end
            end

            for (e = findends),          % try to match the end of the curve
                  if ((any(ismember(currintersectend, endpt(e, 3:4)))) &...
                              (any(ismember(currdeltaend, endpt(e, 1)))))
                        curveref(idxcrv, 2) = e ;
                        break ;
                  elseif ((any(ismember(currintersectend, endpt(e, 5:6)))) &...
                              (any(ismember(currdeltaend, endpt(e, 2)))))
                        curveref(idxcrv, 2) = e ;
                        break ;
                  end
            end
      end

      % Now work out the order in which to traverse the input points to generate
      % output points in the correct chain order.
      % For an open contour there must be exactly two unset locations (0) in the curveref
      % tables corresponding to first & last curves in the contour (ie. no start/end point).
      % Starting with the first curve, use the connecting curve numbers to traverse the
```

```
% curves in the correct direction and order, looking up start & end scan lines and
% building an indexing table as we go.
%
ncurves = size(curveref, 1) ;
linemapindex = zeros(ncurves, 4) ;
idxpts = 1 ;
curve = 0 ;
from = 0 ;
to = 0 ;
lastcurve = 0 ;

[curve, handed] = find(curveref==lastcurve) ;

% Handle the case of a closed contour - just choose the first in the list
%
if (isempty(curve))
        curve = 1 ;
        handed = 1 ;
end

curve = curve(1) ;
handed = handed(1) ;
if (handed==1)
        starthand = 1 ;
        endhand = 2 ;
else
        starthand = 2 ;
        endhand = 1 ;
end

for (idxcrv = 1:ncurves),
        if (curve==0)
                error = 1 ;
                break ;
        end

        if (curveref(curve, handed)==lastcurve)
                from = starthand ;
                to = endhand ;
        else
                from = endhand ;
                to = starthand ;
        end

        linemapindex(idxpts, 1) = endpt(curve, from) ;
        linemapindex(idxpts, 2) = endpt(curve, to) ;
```

```
        linemapindex(idxpts, 3) = curve ;
                linemapindex(idxpts, 4) = to - from ;
                idxpts = idxpts + 1 ;

                lastcurve = curve ;
                curve = curveref(lastcurve, to) ;
        end

        % Generate the list of reordered points.
        %
        if (error==0) ;
                ncurves = size(curveref, 1) ;
                newindex = 1 ;
                for (idxcrv = 1:ncurves),
                        first = linemapindex(idxcrv, 1) ;
                        last = linemapindex(idxcrv, 2) ;
                        curve  = linemapindex(idxcrv, 3) ;
                        incr  = linemapindex(idxcrv, 4) ;

                        for idxpt = first: incr: last,
                                pointl = linemapl(idxpt, curve) ;
                                pointh = linemaph(idxpt, curve) ;
                                point = pointl + round((pointh-pointl)/2) ;    % choose
middle pt of a run
                                newpoints(newindex, 1) = scans(idxpt) ;
                                newpoints(newindex, 2) = point ;
                                newindex = newindex + 1 ;
                        end
                end
        end
```

Figure 6

```
function writebezier(fid, cntr_points, cntr_level, cntr_type, cntr_cols, max_levels, max_fit,
min_component_len, bezerror)
%
% Fit Bezier curves to each of the labelled components using
% adaptive piecewise cubic Bezier fitting algorithm (Graphics Gems P.612)
% Write the SVG file.
% Paint filled regions back-to-front; then holes front-to-back.
%

% Paint filled regions back-to-front
%
for level = 1:max_levels,
        idx_contours = find(((cntr_level(:))'==level) & ((cntr_type(:))'==1)) ;
        if (~isempty(idx_contours))
                for ic = idx_contours,
                        if (~isempty(cntr_points{ic}))
                                points = (cntr_points{ic})' ;           % make into 2-by-n
matrix

                                % The points are still 8-connected - not necessary for curve
fitting.
                                % (also, having too many points seems to break the program).
                                % 'Thin' them out so as to have approx constant number for
any size contour.
                                %
                                npoints = length(points) ;
                                incr = 1 + npoints/max_fit ;
                                fitinds = round(1:incr:npoints) ;
                                points = points(:, fitinds) ;
                                npoints = length(points) ;                  % recalculate
length of points matrix

                                if (npoints > min_component_len)
                                        SVGgroupStart(fid, cntr_cols(ic), 1, cntr_cols(ic),
level) ;
                                        SVGpathStart(fid) ;          % Write SVG block
start

                                        % Piecewise cubic Bezier fitting algorithm.
                                        %
                                        [degree, bezsections] = FitCurves(points, npoints,
bezerror) ;

                                        ncurves = length(degree) ;
                                        idxbez = 1 ;
```

```
                                nctrl = degree(1) + 1 ;
                                bezcurve = round(bezsections(:, idxbez:(idxbez+nctrl-
1))) ;
                                SVGmoveto(fid, bezcurve(:, 1)) ;        % Start new
curve

                                % write the SVG data
                                %
                                for i = 1:ncurves,
                                        nctrl = degree(i) + 1 ;
                                        bezcurve = round(bezsections(:,
idxbez:(idxbez+nctrl-1))) ;
                                        SVGwritepath(fid, bezcurve, degree(i)) ;
                                        idxbez = idxbez + nctrl ;
                                end
                                SVGpathEnd(fid) ;               % Write SVG block
end
                                SVGgroupEnd(fid) ;
                        end
                end
        end
end


% Paint holes  front-to-back
%
for level = max_levels:-1:1,
        idx_contours = find(((cntr_level(:))'==level) & ((cntr_type(:))'==0)) ;
        if (~isempty(idx_contours))
                for ic = idx_contours,
                        points = (cntr_points{ic})' ;           % make into 2-by-n matrix


                        % Thin
                        %
                        npoints = length(points) ;
                        incr = 1 + npoints/max_fit ;
                        fitinds = round(1:incr:npoints) ;
                        points = points(:, fitinds) ;
                        npoints = length(points) ;                      % recalculate length of
points matrix

                        if (npoints > min_component_len)
                                SVGgroupStart(fid, cntr_cols(ic), 1, cntr_cols(ic), level) ;
                                SVGpathStart(fid) ;
```

```
                    [degree, bezsections] = FitCurves(points, npoints, bezerror) ;
                    ncurves = length(degree) ;
                    idxbez = 1 ;

                    nctrl = degree(1) + 1 ;
                    bezcurve = round(bezsections(:, idxbez:(idxbez+nctrl-1))) ;
                    SVGmoveto(fid, bezcurve(:, 1)) ;

                    % write the SVG data
                    %
                    for i = 1:ncurves,
                            nctrl = degree(i) + 1 ;
                            bezcurve = round(bezsections(:, idxbez:(idxbez+nctrl-
1))) ;
                            SVGwritepath(fid, bezcurve, degree(i)) ;
                            idxbez = idxbez + nctrl ;
                    end
                    SVGpathEnd(fid) ;
                    SVGgroupEnd(fid) ;
                end
            end
        end
end
```

Figure 7

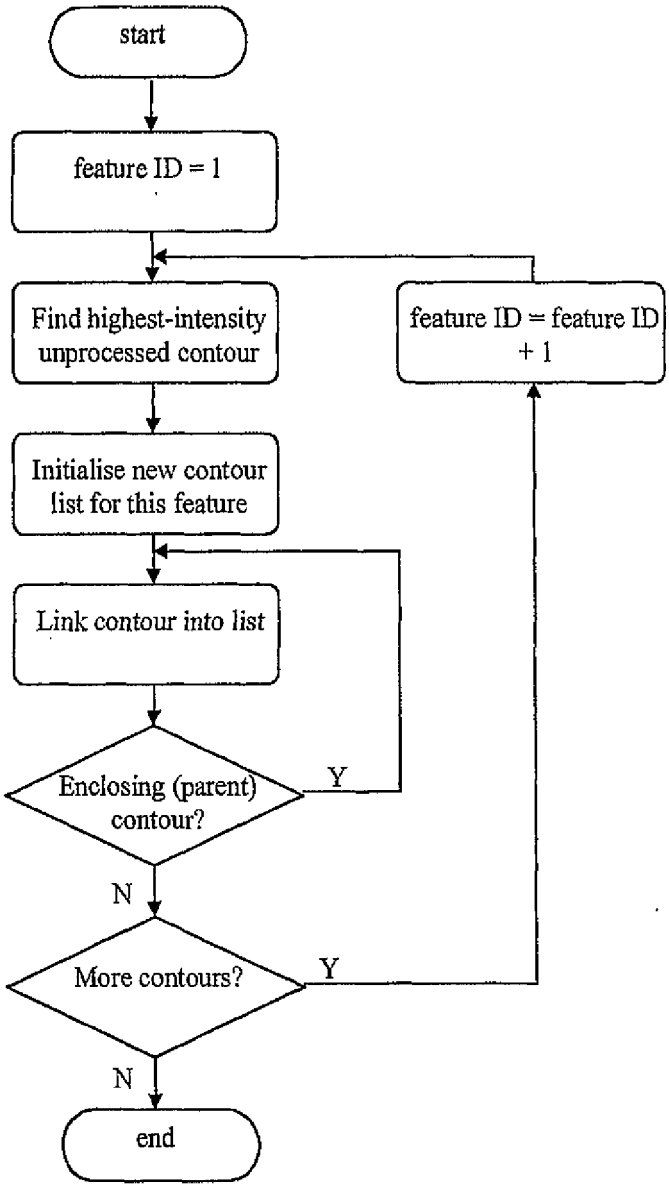

start
feature ID = 1
Find highest-intensity unprocessed contour
feature ID = feature ID + 1
Initialise new contour list for this feature
Link contour into list
Enclosing (parent) contour?
Y
N
More contours?
Y
N
end

Figure 8

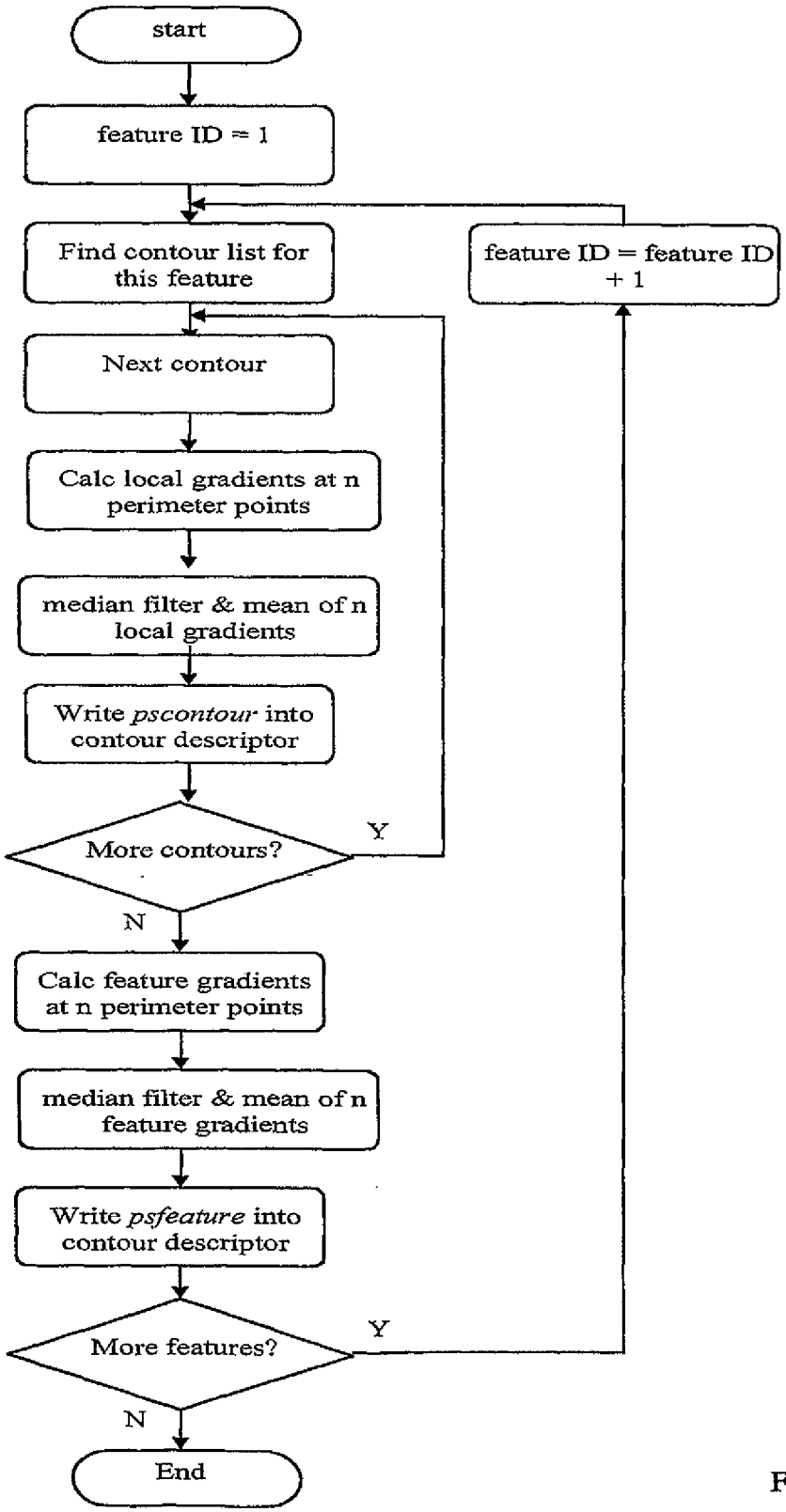

start
feature ID = 1
Find contour list for this feature
feature ID = feature ID + 1
Next contour
Calc local gradients at n perimeter points
median filter & mean of n local gradients
Write *pscontour* into contour descriptor
More contours?
Y
N
Calc feature gradients at n perimeter points
median filter & mean of n feature gradients
Write *psfeature* into contour descriptor
More features?
Y
N
End

Figure 9

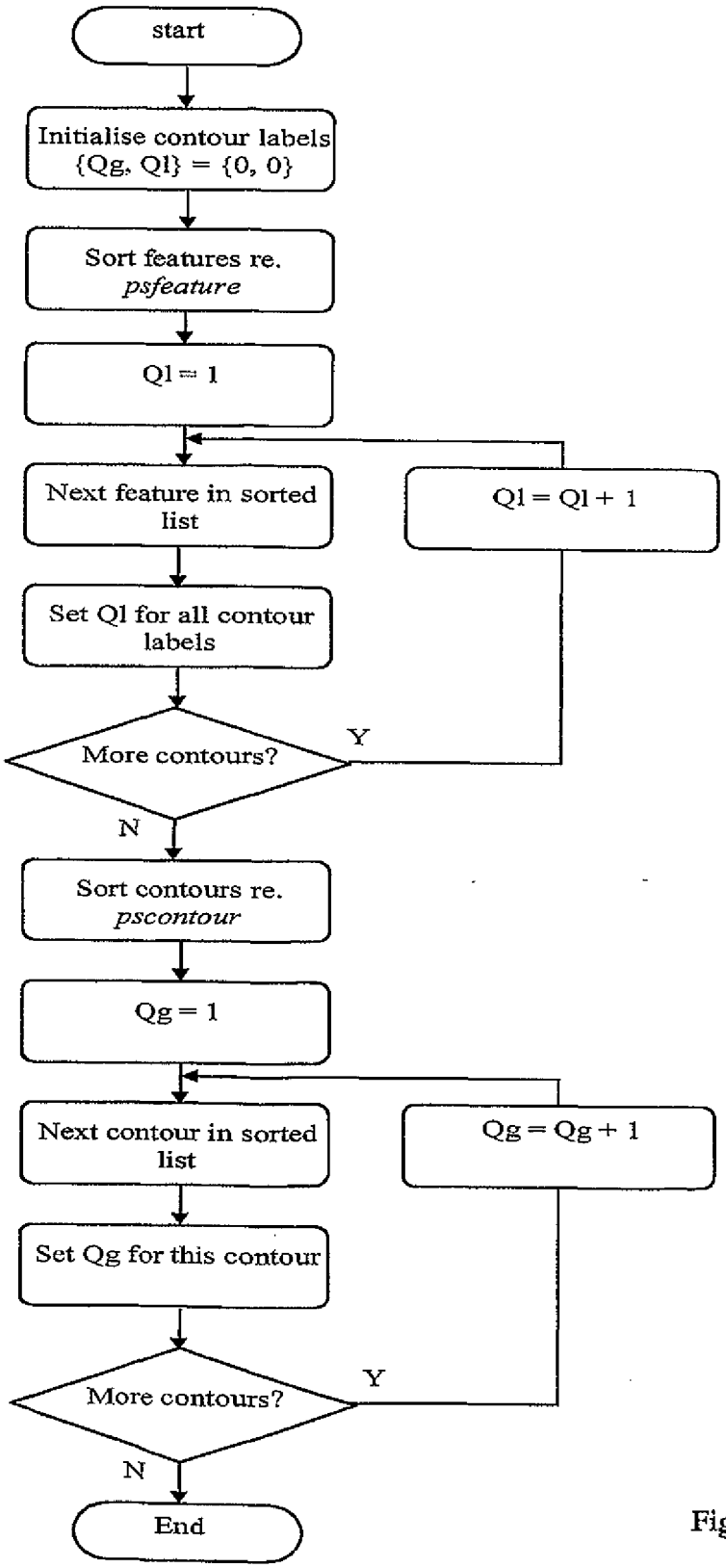

start
Initialise contour labels {Qg, Ql} = {0, 0}
Sort features re. psfeature
Ql = 1
Next feature in sorted list
Ql = Ql + 1
Set Ql for all contour labels
More contours?
Y
N
Sort contours re. pscontour
Qg = 1
Next contour in sorted list
Qg = Qg + 1
Set Qg for this contour
More contours?
Y
N
End

Figure 10

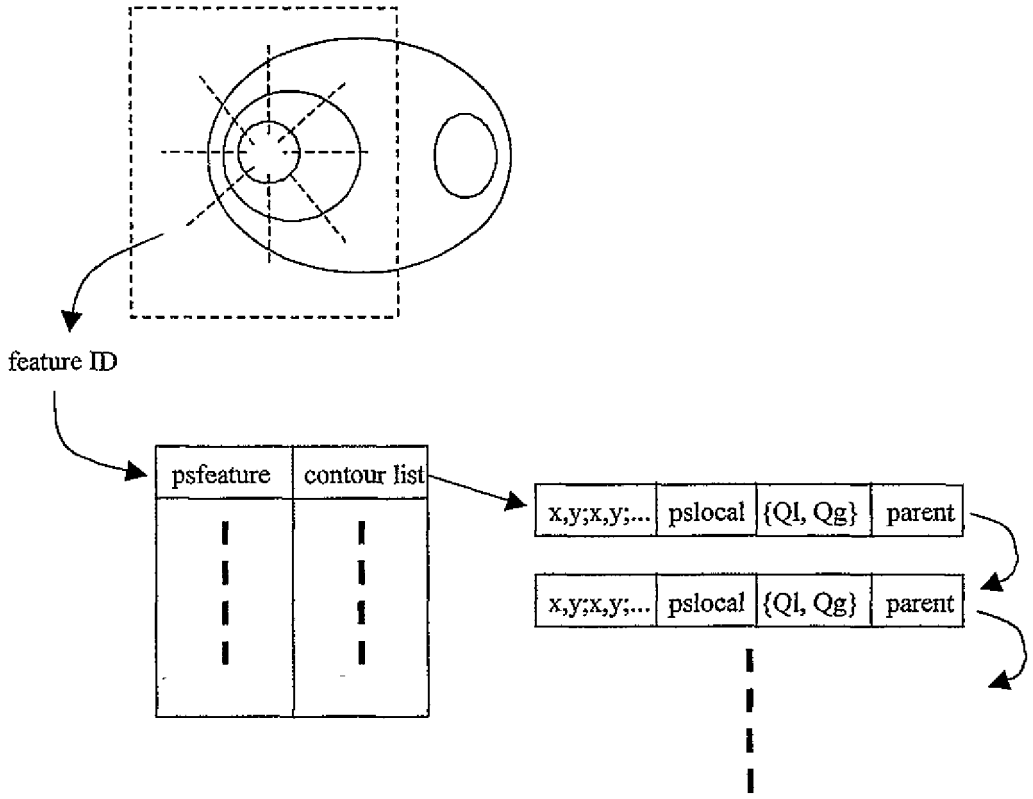

feature ID
psfeature
contour list
x,y;x,y;...
pslocal
{Ql, Qg}
parent
x,y;x,y;...
pslocal
{Ql, Qg}
parent

Figure 11

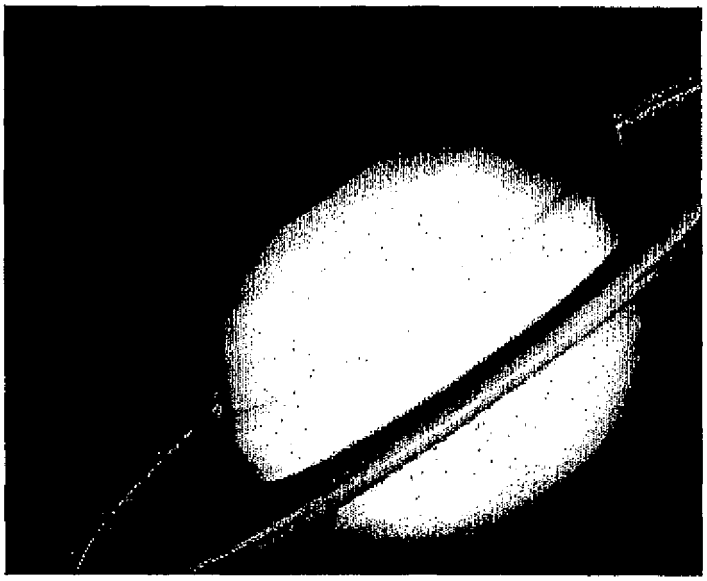


Figure 12

Figure 13

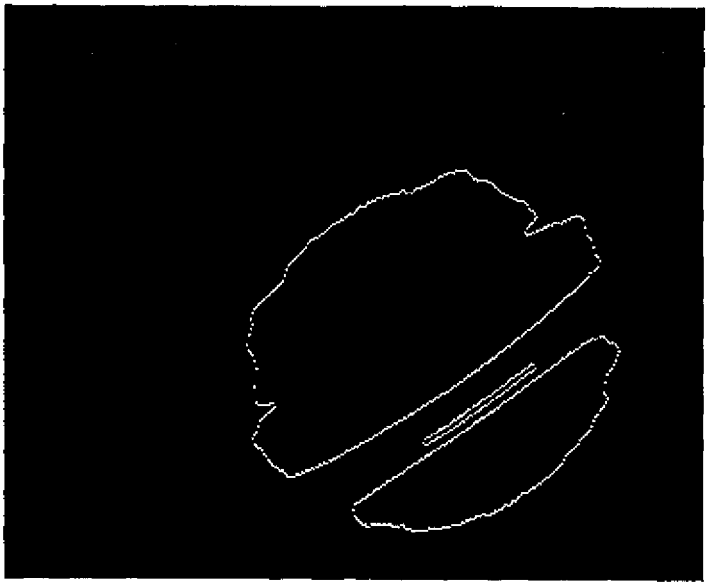


Figure 14

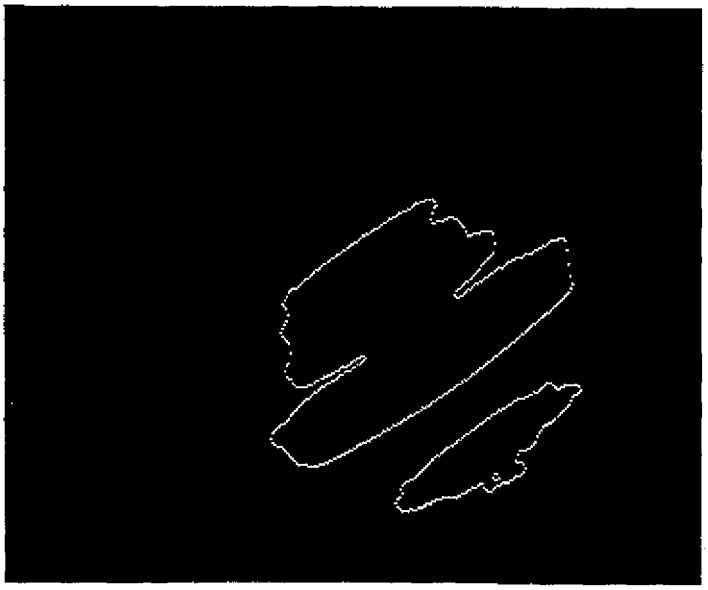


Figure 15

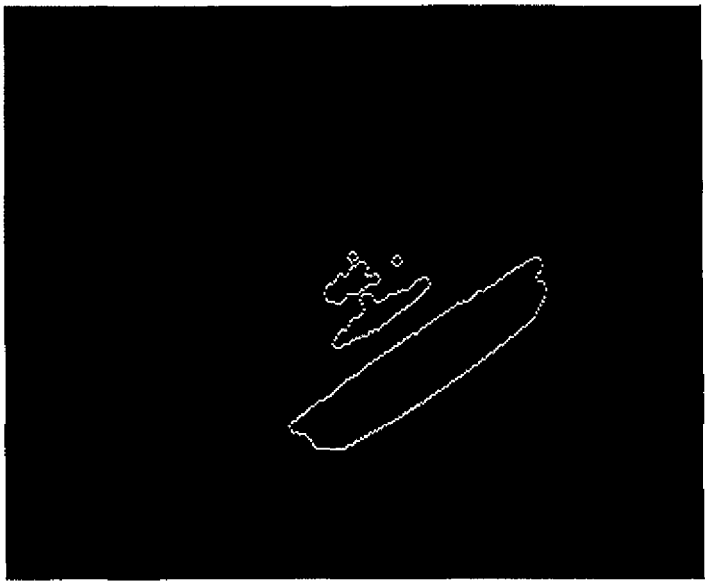


Figure 16

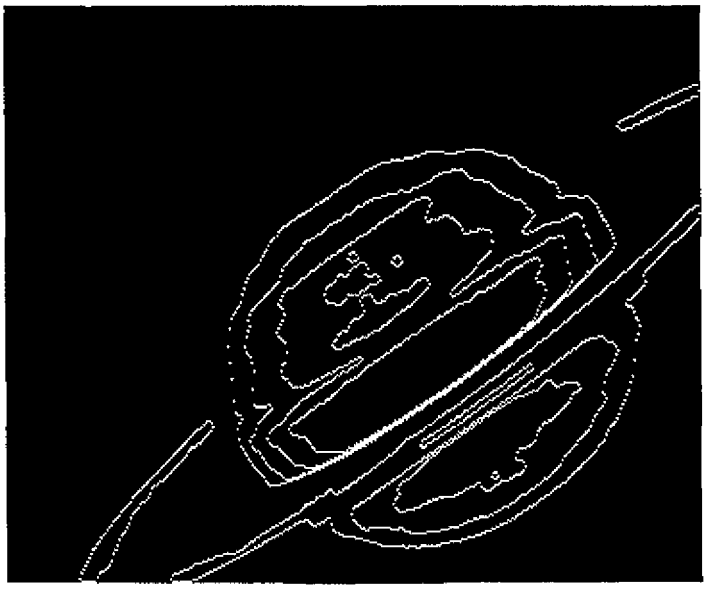


Figure 17

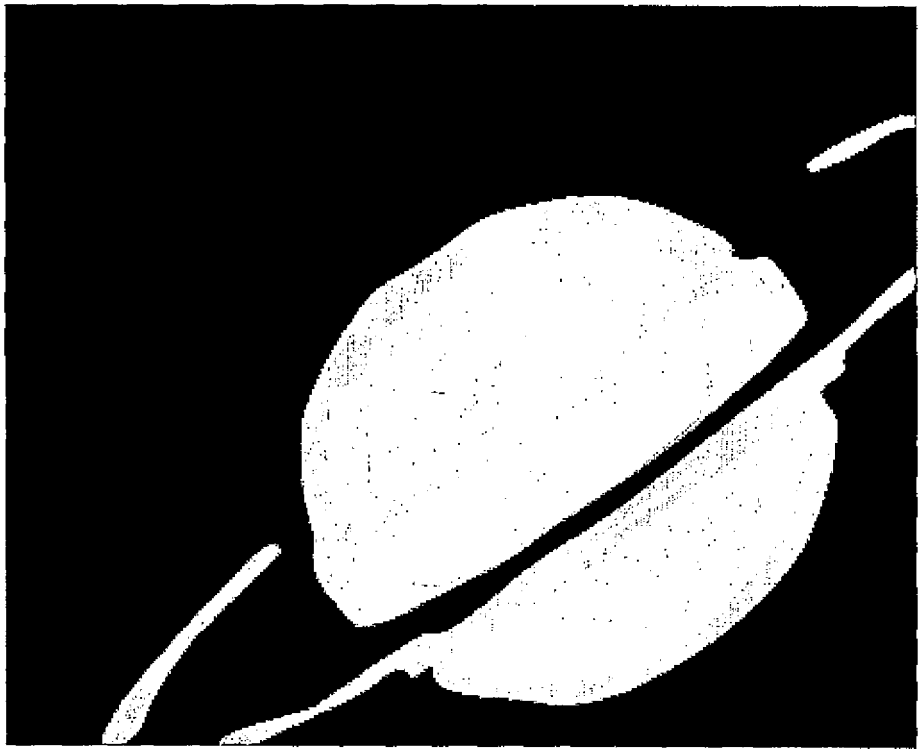


Figure 18

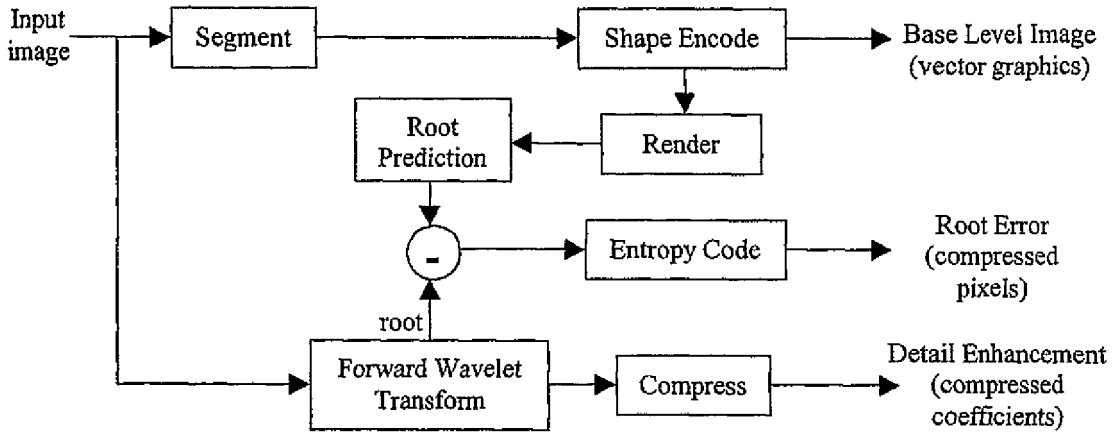


Figure 19

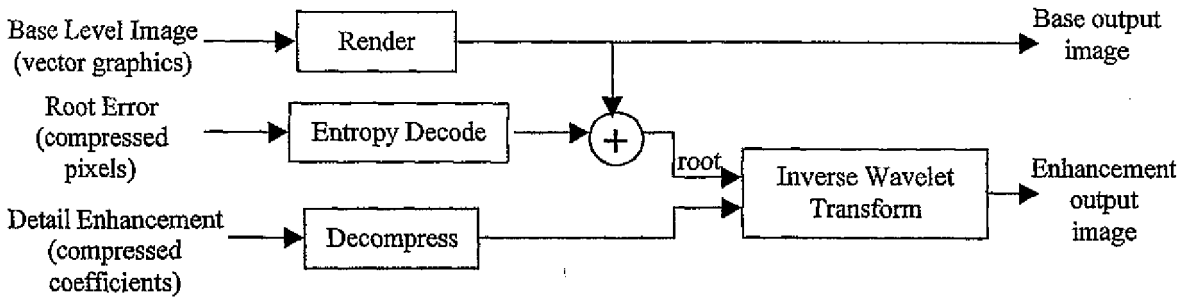


Figure 20